

Agenda: Service Fabric and Microservices

- What are Microservices
- Overview of Service Fabric
- How Service Fabric Works
- Key capabilities of Service Fabric
- Cloud Services vs Service Fabric
- Service Fabric Application Model
- Service Fabric Programming Model
- Prepare development environment
- Create a .NET Service Fabric Application
- Making the service to listen and connect with other services
- Specifying environment specific parameters
- Deploying Service Fabric application to Azure
- Service Fabric Explorer
- Scale applications and services in a cluster.
- Perform a rolling application upgrade
- Guest Executables programming model

What are Microservices

Microservice applications are composed of small, independently versioned, and scalable customer-focused services that communicate with each other over standard protocols with well-defined interfaces.

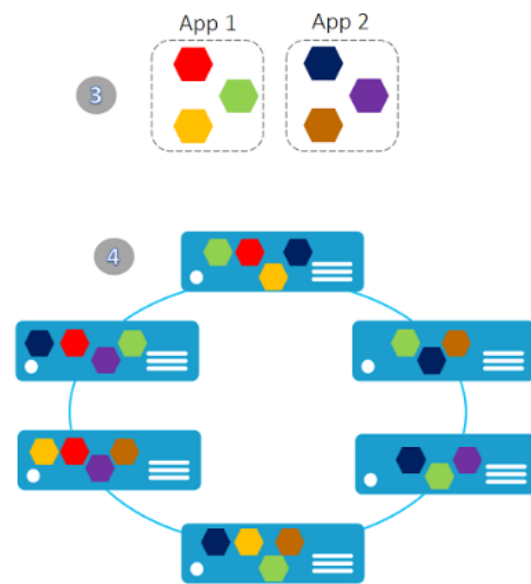
The microservice approach is to compose your application of many small services.

Comparison between application development approaches:

Monolithic application approach



Microservices application approach



1) A monolithic app contains domain-specific functionality and is normally divided by functional layers, such as web, business, and data.

2) You scale a monolithic app by cloning it on multiple servers/virtual machines/containers.

3) A microservice application separates functionality into separate smaller services.

4) The microservices approach scales out by deploying each service independently, creating instances of these services across servers/virtual machines/containers.

The services run in containers that are deployed across a cluster of machines. Smaller teams develop a service that focuses on a scenario and independently test, version, deploy, and scale each service so that the entire application can evolve.

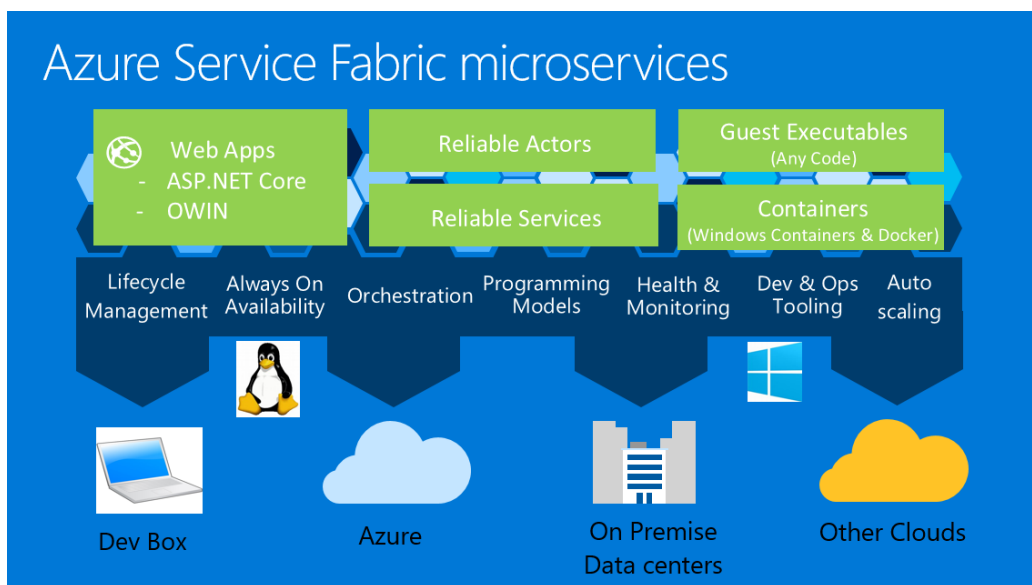
Characteristics of Microservices:

- Encapsulate a customer or business scenario. Code solving a domain-specific problem?
- Developed by a small engineering team using a particular technology stack.
- Written in any programming language and use any framework.
- Consist of code and (optionally) state, both of which are independently versioned, deployed, and scaled.
- Interact with other microservices over well-defined interfaces and protocols.
- Can be upgraded independently of calling services.
- Needs to have a unique name so that its current location is discoverable.
- A microservice needs to be resilient to failures and restart often on another machine for availability reasons.

Overview of Azure Service Fabric

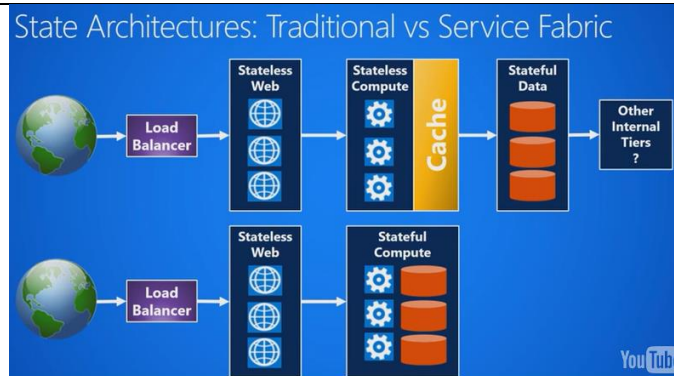
<https://channel9.msdn.com/Blogs/Azure/Azure-Service-Fabric>

- **Service fabric is a platform** that provides you with a lot of built in tools, mechanisms, APIs to solve many of the problems you will encounter when building **distributed cloud based** applications.
- Service Fabric is a microservices platform that gives every **microservice a unique name** that can be either **stateless or stateful**. It provides a sophisticated, lightweight runtime to build distributed, scalable and reliable microservices that run at high density on a **shared pool of machines**, which is referred to as a **cluster**.
- Service Fabric is tailored to create cloud native services that can start small, as needed, and grow to massive scale with hundreds or thousands of machines.
- Service Fabric application lifecycle management capabilities enable application developers and administrators to avoid complex infrastructure problems and focus on implementing mission-critical, demanding workloads that are scalable, reliable, and manageable.
- Service Fabric powers many Microsoft services today, including Azure SQL Database, Azure Cosmos DB, Cortana, Microsoft Power BI, Microsoft Intune, Azure Event Hubs, Azure IoT Hub, Dynamics 365, Skype for Business, and many core Azure services.
- Today's Internet-scale services are built using microservices. Examples of microservices include protocol gateways, user profiles, shopping carts, inventory processing, queues, and caches.
- Service Fabric can be run in Azure, on-premises or in alternative cloud provider sites and you can create clusters for Service Fabric in many environments, including Azure or on premises, on Windows Server, or on Linux.

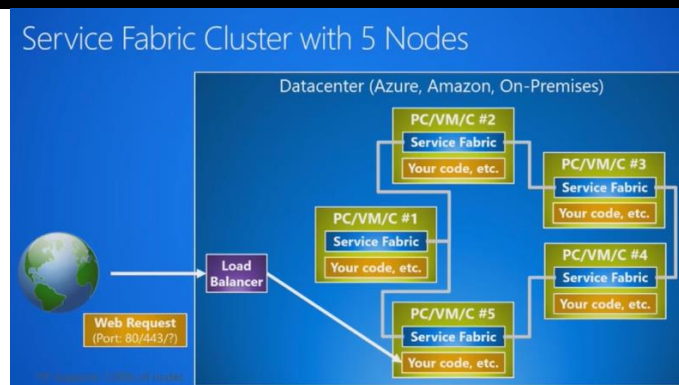


Stateless and Stateful microservices for Service Fabric

- Stateless microservices (such as protocol gateways and web proxies) do not maintain a mutable state outside a request and its response from the service. Azure Cloud Services worker roles are an example of a stateless service.
- Stateful microservices (such as user accounts, databases, devices, shopping carts, and queues) maintain a mutable, authoritative state beyond the request and its response.



How Azure Service Fabric Works



1. In a data center, we will provision a bunch of Computers or Virtual Machines called as Nodes. Images shows 5 of them.
2. Each of them will have an OS installed on them and we need to install Service Fabric Components on each one of them.
3. We need to deploy **cluster manifest file** which has endpoints of all the machines so that service fabric components know how to communicate with service fabric component on the other machines. When these machines are talking to each other, that's when we call that as a cluster. (Image shows cluster of 5 machines)
4. Then we need to setup Load Balancer. This is provided by Azure in Azure Data Center because Service Fabric will be running on top of the **VM Scale Set feature** that's inside Azure that integrates with Load Balancer.
5. The Software Developer then will package the code which we would deploy in the cluster. The code package (Application Type Package) through the load balancer is then uploaded to one of the machines in the service fabric and then service fabric will end up copying that code to all the machines in the cluster. So now our code is deployed to all of the nodes running within the cluster. Service Fabric is letting us treat the entire cluster as if it is one machine.
6. On the web request from the client, it goes into the load balancer then to one of the nodes in the cluster which is then forwarded to your code running in that node.
7. The code will then return typically, HTML, JSON, XML etc...

Each Node in Service Fabric has the following components running over it

1. FabricHost.exe
 - a. Auto-Starts at boot.
2. Fabric.exe
 - a. Started by FabricHost.exe.
 - b. Is responsible for Inter-node communication.
3. FabricGateway.exe
 - a. Started by FabricHost.exe.
 - b. Cluster (Intra-node) communication.
4. Your App's Services
 - a. Eg.ASP.NET or other .exe.
 - b. Exposes Public endpoints.

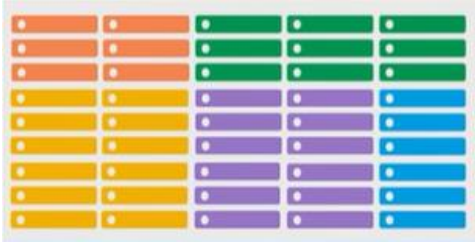
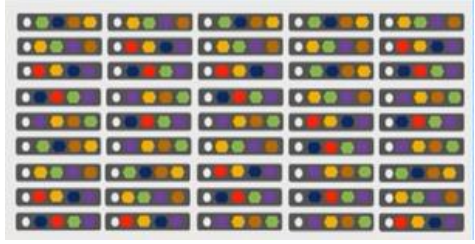
Note: Usually there is a One to One relationship between Node and PC. But on **OneBox** scenario we can have multiple nodes on one PC and we will then have multiple instances of Fabric.exe, FabricGateway.exe and App's Services.

Key capabilities of Service Fabric

By using Service Fabric, you can:

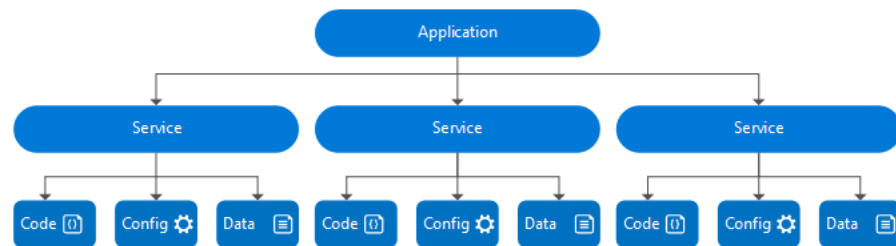
- Develop **scalable applications** that are composed of microservices by using the Service Fabric programming models, containers, or any code.
- Develop **highly reliable** stateless and stateful microservices. Simplify the design of your application by using **stateful microservices**.
- Use the novel Reliable Actors programming model to create cloud objects with self-contained code and state.
- Deploy to Azure or to on-premises datacenters that run Windows or Linux with zero code changes. Write once, and then deploy anywhere to any Service Fabric cluster.
- Deploy applications in seconds, at high density with hundreds or thousands of applications per machine thus reducing the cost as unlike in Cloud Service where each role would be deployed in a separate VM.
- Deploy different versions of the same application side by side, and upgrade each application independently.
- Manage the lifecycle of your applications without any downtime, including breaking and nonbreaking upgrades.
- Scale out or scale in the number of nodes in a cluster. As you scale nodes, your applications automatically scale.
- Monitor and diagnose the health of your applications and set policies for performing automatic repairs.
- Service Fabric recovers from failures and optimizes the distribution of load based on available resources.

Cloud Services vs Service Fabric

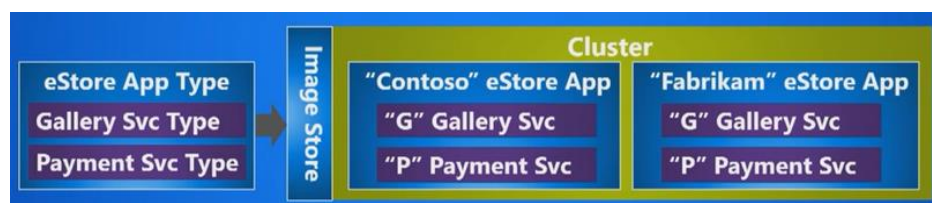
Cloud Services	Service Fabric
	
Each role/instance per VM	Many Service instances share a PC/VM
Slow deployment & upgrades	Fast deployment & upgrades
Slow to scale role instances up/down	Fast to scale service instances up/down
Emulator for development	One Box cluster for development.

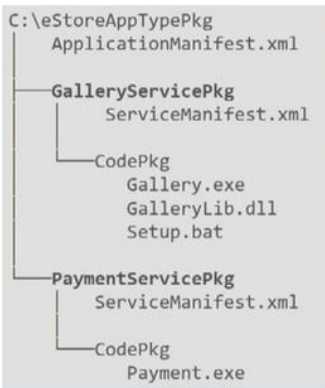
Service Fabric Application Model

- An application is a collection of constituent services that perform a certain function or functions.
- A service performs a complete and standalone function and can start and run independently of other services.
- A service is composed of code, configuration, and data. For each service, code consists of the executable binaries, configuration consists of service settings that can be loaded at run time, and data consists of arbitrary static data to be consumed by the service.



- An **application type** is a categorization of an application and consists of a collection of service types.
- A **service type** is a categorization of a service.
- The categorization can have different settings and configurations, but the core functionality remains the same.
- An **instance** of Application type is deployed in the cluster and its then called **Named Application**.
- The **instances of a service** are the different service configuration variations of the same service type.
- The code for different application instances will run as **separate processes** even when hosted by the same Service Fabric node. Furthermore, the lifecycle of each application instance can be managed (i.e. upgraded) independently.
-



Service Package Directory and Service Manifest:

Two manifest files are used to describe applications and services:

1. Application Manifest.
2. Service Manifest

Service Fabric Programming Model

1. Stateless Reliable Services
2. Stateful Reliable Services
3. Guest Executables
4. Reliable Actors Service
5. ASP.NET Core

Prepare development environment**To use Visual Studio 2017**

Service Fabric Tools are part of the Azure Development and Management workload in Visual Studio 2017. Enable this workload as part of your Visual Studio installation. In addition, you need to install the **Microsoft Azure Service Fabric SDK**, using Web Platform Installer.

- [Install the Microsoft Azure Service Fabric SDK](#)

To use Visual Studio 2015 (requires Visual Studio 2015 Update 2 or later)

For Visual Studio 2015, Service Fabric tools are installed together with the SDK, using the Web Platform Installer:

- [Install the Microsoft Azure Service Fabric SDK and Tools](#)

Enable PowerShell script execution:

Service Fabric uses Windows PowerShell scripts for creating a local development cluster and for deploying applications from Visual Studio. By default, Windows blocks these scripts from running. To enable them, you must modify your PowerShell execution policy. Open PowerShell as an administrator and enter the following command:

[Set-ExecutionPolicy](#) -ExecutionPolicy **Unrestricted** -Force -Scope CurrentUser

Create a .NET Service Fabric application

Using this application, you learn how to:

- Create an application using .NET and Service Fabric
- Use ASP.NET core as a web front-end
- Store application data in a stateful service
- Debug your application locally
- Deploy the application to a cluster in Azure
- Scale-out the application across multiple nodes
- Perform a rolling application upgrade

Walkthrough the sample application:

1. **Launch Visual Studio as an Administrator.**
2. File → New → Project → Cloud > Service Fabric Application.
3. **Name=MyFabricApplication → OK**
4. Choose **Stateless ASP.NET Core**, and name your service **MyStatelessWebApp** → Choose Web Application (Model View Cotroller) Template → OK

The Service Fabric application project – MyFabricApplication

This project does not contain any code directly. Instead, it references a set of service projects. In addition, it contains other types of content to specify how your application is composed and deployed. It contains:

- A reference to the Web App project - MyWebFrontEndWebApp
- **ApplicationPackageRoot** folder containing the **ApplicationManifest.xml** file describing your Service Fabric application
- **ApplicationParameters** folder containing deployment parameters for local (Local.1Node.xml and Local.5Node.xml) and cloud (Cloud.xml) deployments.
- **PublishProfiles** containing deployment profiles for local (Local.1Node.xml and Local.5Node.xml) and cloud (Cloud.xml) deployments. The Cloud profile is used to publish to Azure
- **Scripts** containing the scripts used for deploying the application to the cluster
- **Packages.config** used to indicate the packages associated with this application

The service project – MyStatelessWebApp

This project is your ASP.NET Core Web Application project, containing the code and configuration for your service. There are no specific requirements to the code you write as part of your controllers, when running an ASP.NET Core Web API as a reliable service in Service Fabric.

- **Controllers** folder containing the controllers for this project.
- **PackageRoot** folder containing the service configuration and ServiceManifest.xml.
- **Program.cs** which is the host executable of the stateless service.

- **ServiceEventSource.cs** contains the structured events that can be viewed in the Diagnostic Events window and can be captured by Azure diagnostics.
- **Startup.cs** containing the application server startup configuration.
- **MyStatelessWebApp.cs** contains the classed implementing the stateless service.

At this point, you have a functioning service that can be hosted within Service Fabric. **Press F5** to see the service running. Within Visual Studio, the **Diagnostic Events panel** will be visible and display messages coming from within the application.

Add a stateful back-end service to your application

5. Solution Explorer → **MyFabricApplication Project** → right click on **Services** → **Add** → **New Service Fabric Service**.
6. In the New Service Fabric Service dialog, **choose Stateful Service**, and name the service **MyStatefulService** → **OK**.

Note: Once your service project is created, you have two services in your application. As you continue to build your application, you can add more services in the same way. Each can be independently versioned and upgraded.

7. **Press F5** to debug the application.
8. Open **Diagnostics Event Viewer (View, Other Windows)**, where you can see trace-output from your services.

The Diagnostics Events viewer shows you trace messages from all services that are part of the Visual Studio solution being debugged. By pausing the Diagnostics Event viewer, you are able to expand one of the service messages, to inspect its properties. By doing so, you can see which service in the cluster emitted the message, which node that service instance is currently running on and other information.

Making the Service to Listen and connect with other Services

Steps to have your Service Listen

1. For input endpoint(s), configure **load balancer** to allow traffic over desired port(s)
2. Add endpoint(s) to ServiceManifest.xml
3. When service starts, listen for the network traffic
 - a) Override **CreateServiceInstanceListener** and return methods that create listener objects/endpoints names.
 - b) When service instance starts, SF invokes methods to create listener objects
 - c) Each listener object starts listening (tcp, http) & returns endpoint to SF.
 - SF runtime sends all names/endpoints to SF **naming service**
 - Endpoints are visible in Service Fabric Explorer

Continue with Walkthrough:

9. Solution Explorer → Solution → Right Click → Add → New Project
10. Visual C# → Class Library (.NET Framework) template → Name = MySharedClassLibrary, Ensure that .NET Framework = 4.5.2
11. Right-click the solution in the Solution Explorer, and choose **Manage NuGet Packages for Solution**.
12. Choose **Browse** and search for **Microsoft.ServiceFabric.Services.Remoting**. Choose to install it for the three projects in the solution (Except the Fabric Application Project).

Note: The version of referenced library should be same as the version of **Microsoft.ServiceFabric.Services** which is already referenced in the Stateful project. Stateful Project → **packages.config** → Note the version of **Microsoft.ServiceFabric.Services**.

13. In ClassLibrary project add the interface as below:

```
using Microsoft.ServiceFabric.Services.Remoting;

public interface ICounter : IService
{
    Task<long> GetCounterAsync();
}
```

14. Right click the ClassLibrary project in Solution Explorer and select Properties. Select the Build tab, then select the **x64** value in the Platform target dropdown

Implement the interface in your Stateful Service

15. In MyStatefulService, add a reference to the class library project (MySharedClassLibrary) that contains the interface.
16. Open MyStatefulService.cs file and extend it to implement, **ICounter** interface
17. Implement the method GetCounter as below:

```
public async Task<long> GetCounterAsync()
{
    using (var tx = this.StateManager.CreateTransaction())
    {
        var myDictionary = await this.StateManager.GetOrAddAsync<IReliableDictionary<string, long>>("myDictionary");
        var result = await myDictionary.TryGetValueAsync(tx, "Counter");
        if (result.HasValue)
            return result.Value;
        else
            return 0;
    }
}
```

```
}
```

18. Replace the overridden method **CreateServiceReplicaListeners**

Note: With this method, you can specify one or more communication listeners, based on the type of communication that you want to enable for your service.

Note:

- In this case, we replace the existing `CreateServiceReplicaListeners` method and provide an instance of `ServiceRemotingListener`, which creates an RPC endpoint that is callable from clients through `ServiceProxy`.
- The `Microsoft.ServiceFabric.Services.Remoting.Runtime` namespace contains an extension method, **CreateServiceRemotingListener** for both stateless and stateful services that can be used to create a remoting listener using the default remoting transport protocol.

```
using Microsoft.ServiceFabric.Services.Remoting.Runtime;

protected override IEnumerable<ServiceReplicaListener> CreateServiceReplicaListeners()
{
    //return new ServiceReplicaListener[0];
    return new List<ServiceReplicaListener>()
    {
        new ServiceReplicaListener(
            (context) =>
                this.CreateServiceRemotingListener(context))
    };
}
```

Call the stateful back-end service from the front-end service

Calling methods on a service by using the remoting stack is done by using a local proxy to the service through the `Microsoft.ServiceFabric.Services.Remoting.Client.ServiceProxy` class. The `ServiceProxy` method creates a local proxy by using the same interface that the service implements.

19. In **MyStatelessWebApp**, Add reference to `MySharedClassLibrary`.

20. Add the following code `HomeController`

```
public async Task<ActionResult> Index()
{
    ICounter objCounter = ServiceProxy.Create<ICounter>(new
Uri("fabric:/MyFabricApplication/MyStatefulService"), new ServicePartitionKey(0));
    ViewBag.count = await objCounter.GetCounterAsync();
    return View();
}
```

21. Edit Views\Home\Index.cshtml as below

Current Counter: @ViewBag.count

22. Press F5 to debug the application and observe the output.

WCF Based Communication Stack for Reliable service:

<https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-reliable-services-communication-wcf>

Specifying environment-specific parameters

Per-environment service configuration settings:

The Service Fabric application model enables services to include configuration packages that contain **custom key-value pairs** that are readable at run time. The values of these settings can also be differentiated by environment by specifying a **ConfigOverride** in the application manifest.

23. Edit file **MyStatefulService**\PackageRoot\Config\Settings.xml and add the following section.

```
<Section Name="MyConfigSection">
  <Parameter Name="Key1" Value="Value1" />
</Section>
```

24. To read the above configuration, add the following to MyStatefulService (Stateful) service **constructor**

```
var configPackage = context.CodePackageActivationContext.GetConfigurationPackageObject("Config");
var key1 = configPackage.Settings.Sections["MyConfigSection"].Parameters["Key1"].Value;
```

25. In case required, we can override this value for a specific application/environment pair, Edit the file

MyFabricApplication\ApplicationPackageRoot\ApplicationManifest.xml

```
<Parameters>
  ...
  <Parameter Name="MyStatefulService_Key1" DefaultValue="DefaultKey1" />
</Parameters>

<ServiceManifestImport>
  <ServiceManifestRef ServiceManifestName="MyStatefulServicePkg" ServiceManifestVersion="1.0.0" />
  <ConfigOverrides>
    <ConfigOverride Name="Config">
      <Settings>
        <Section Name="MyConfigSection">
          <Parameter Name="Key1" Value="[MyStatefulService_Key1]" />
        </Section>
      </Settings>
    </ConfigOverride>
  </ConfigOverrides>
</ServiceManifestImport>
```

```

</ConfigOverride>
</ConfigOverrides>
</ServiceManifestImport>

```

26. Value of Parameter can now be mentioned in MyFabricApplication\ApplicationParameters\

1. Cloud.xml
2. Local.1Node.xml
3. Local.1Node.xml

```

<Parameters>
...
<Parameter Name="MyStatefulService_Key1" Value="ValueFromXML" />
</Parameters>

```

Setting and using environment variables:

You can specify and set environment variables in the **ServiceManifest.xml** file and then override these in the ApplicationManifest.xml file on a per instance basis.

27. Edit MyStatefulService\PackageRoot\ServiceManifest.xml

```

<CodePackage Name="Code" Version="1.0.0">
  <EntryPoint>
    ...
  </EntryPoint>
  <EnvironmentVariables>
    <EnvironmentVariable Name="Key2" Value="value2"/>
  </EnvironmentVariables>
</CodePackage>

```

28. To override the environment variables in the **ApplicationManifest.xml**, reference the code package in the ServiceManifest with the `EnvironmentOverrides` element.

```

<ServiceManifestImport>
  <ServiceManifestRef ServiceManifestName="MyStatefulServicePkg" ServiceManifestVersion="1.0.0" />
  <EnvironmentOverrides CodePackageRef="Code">
    <EnvironmentVariable Name="Key2" Value="ValueInAppManifest"/>
  </EnvironmentOverrides>
</ServiceManifestImport>

```

30. Once the named service instance is created you can access the environment variables from code. e.g. In C# you can do the following

```
var key2 = Environment.GetEnvironmentVariable("Key2");
```

Deploy the application to Azure

28. Azure Portal → More Services → Service Fabric Cluster → Add Service Fabric Cluster
29. Fill the Service Fabric Basic Form, Cluster name=MyFabricApplicationCluster, The user name and password used to log in to the virtual machine, Ideally create a New Resource Group.
30. Fill out the **Cluster configuration** form, Node type count=1,

Node types define the VM size, number of VMs, custom endpoints, and other settings for the VMs of that type. Each node type defined is setup has a separate virtual machine scale set, which is used to deploy and managed virtual machines as a set.

Each node type can be scaled up or down independently, have different sets of ports open, and can have different capacity metrics.

The first, or primary, node type is where Service Fabric system services are hosted and must have five or more VMs as this is the node type where Service Fabric system services are placed.

A common scenario for multiple node types is an application that contains a front-end service and a back-end service. You want to put the front-end service on smaller VMs (VM sizes like D2) with ports open to the Internet, but you want to put the back-end service on larger VMs (with VM sizes like D4, D6, D15, and so on) with no Internet-facing ports open.

31. Select **Configure each node type** and fill out the **Node type configuration** form.
32. **Configure custom endpoints = 80.**
This field allows you to enter a **comma-separated list of ports** that you want to expose through the **Azure Load Balancer** to the public Internet for your applications. For example, if you plan to deploy a web application to your cluster, enter "80" here to allow traffic on port 80 into your cluster
33. In the **Cluster configuration** form, set **Diagnostics** to **On**. In **Fabric version**, select **Automatic** upgrade mode so that Microsoft automatically updates the version of the fabric code running the cluster.
34. Fill out the **Security** form. For this quick start select **Unsecure**. It is highly recommended to create a secure cluster for production workloads since anyone can anonymously connect to an unsecure cluster and perform management operations.
35. Review the summary, Select **Create** to create the cluster.

Visualize the cluster using Service Fabric explorer

Service Fabric Explorer is a good tool for visualizing your cluster and managing applications.

Service Fabric Explorer is a service that runs in the cluster.

The cluster dashboard provides an overview of your cluster, including a summary of application and node health.

The node view shows the physical layout of the cluster. For a given node, you can inspect which applications have code deployed on that node.

- Access it using a web browser by clicking the **Explorer** link of the cluster **Overview** page in the portal:
- You can also enter the address directly into the browser:

<http://MyFabricApplicationCluster.westus.cloudapp.azure.com:19080/Explorer>

Deploy the application in Azure Portal using Visual Studio

36. In Solution Explorer, open up **MyStatelessWebApp** → PackageRoot → **ServiceManifest.xml**.

37. Change the **Port** attribute of the existing Endpoint element to **80** and save your changes.

38. Right-click **MyFabricApplication** in the Solution Explorer and choose **Publish**.

39. From the Dropdown Connection EndPoints, select the already created Service Fabric Cluster.

40. Publish.

41. Open a browser and type in the cluster address - for example, `http://`

`MyFabricApplicationCluster.westus.cloudapp.azure.com`.

You should now see the application running in the cluster in Azure.

42. You can disable one of the node in the cluster and refresh the page from top right and note that the service shifts quickly to another active node.

Scale applications and services in a cluster

Service Fabric services can easily be scaled across a cluster to accommodate for a change in the load on the services. You scale a service by changing the number of instances running in the cluster.

To scale the web front-end service, do the following steps:

43. Open Service Fabric Explorer in your cluster

44. Expand Cluster → Applications → MyFabricApplication → fabric:/MyFabricApplication →

fabric:/MyFabricApplication/ **MyStatelessWebApp**, click on the ellipsis (three dots) choose **Scale Service**

45. You can now choose to scale the number of instances of the web front-end service. Change the number to **2** and click **Scale Service**.

46. You can now see that the service has two instances, and in the tree view you see which nodes the instances run on.

Perform a rolling application upgrade

When deploying new updates to your application, Service Fabric rolls out the update in a safe way. **Rolling upgrades** gives you **no downtime** while upgrading as well as automated rollback should errors occur.

47. Open the **MyStatelessWebApp\Views\Home\Index.cshtml** file in Visual Studio and add some new content → Save the file

48. Right Click on MyFabricApplication → Publish → Manifest Versions...

49. Change the Version of **MyStatelessWebAppPkg**, New Version =2.0.0 → Save

50. Check **Update the Application** checkbox.

51. Open Service Fabric Explorer, browse to the cluster address on port 19080.

52. Click on the **Applications** node in the tree view, and then **Upgrades in Progress** tab in the right-hand pane.

You see how the upgrade rolls through the upgrade domains in your cluster, making sure each domain is healthy before proceeding to the next.

Note: Service Fabric makes upgrades safe by waiting **two minutes** after upgrading the service on each node in the cluster. Expect the entire update to take approximately eight minutes.

53. While the upgrade is running, you can still use the application. Because you have two instances of the service running in the cluster, some of your requests may get an upgraded version of the application, while others may still get the old version.

Use the following link if the Upgrade fails:

<https://blogs.msdn.microsoft.com/maheshk/2017/05/24/azure-service-fabric-error-to-allow-it-set-enabledefaultservicesupgrade-to-true/>

Guest Executables Programming Model

1. An arbitrary executable, written in any language, so you can take your existing applications and host them on a Service Fabric cluster.
2. A guest executable can be packaged in an application and hosted alongside other services.
3. Because guest executables do not integrate directly with Service Fabric APIs, they do not benefit from the full set of features the platform offers, such as custom health and load reporting, service endpoint registration, and stateful compute.

```
<ServiceManifest Name="Guest1Pkg"
  Version="1.0.0"
  xmlns="http://schemas.microsoft.com/2011/01/fabric"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ServiceTypes>
    <StatelessServiceType ServiceTypeName="Guest1Type" UseImplicitHost="true" />
  </ServiceTypes>

  <!-- Code package is your service executable. -->
  <CodePackage Name="Code" Version="1.0.0">
    <EntryPoint>
```



```
<ExeHost>  
  <Program>MyConsoleApp.exe</Program>  
  <Arguments></Arguments>  
  <WorkingFolder>Work</WorkingFolder>  
</ExeHost>  
</EntryPoint>  
</CodePackage>  
</ServiceManifest>
```